Exhibit B

Confessions of an operating systems junkie

Val Henson's weblog

All | Operating Systems

TUESDAY SEPTEMBER 21, 2004

Farewell for now As of October 4th, I will be a Linux developer again. After a many months-long search for a Linux job that was better than my Sun job (no small task), I finally accepted an offer from IBM.

I learned a few things over the past few months. Sun is a great working environment; most other companies score much higher on the Dilbert scale. Which explains why I know so many people who left Sun and then came back a few years later (an option I will consider). After deciding I wanted to quit, I stopped doing all the things I didn't want to do - and ended up being **more** productive. Finally, a little patience (and negotiating skill) will go a long way (unfortunately, these are two qualities I am notoriously short on).

I feel lucky to have had the chance to work on ZFS. It really is a groundbreaking new file system - and so easy to use. Now I just hope that it gets ported to Linux soon, so I can use it.

I may start a new weblog; if so, it will be linked to from my home page:

http://www.nmt.edu/~val

After leafing through the stack of computer science papers on my desk, looking for one last systems paper to talk about, I find I can't pick just one. Here for your reading enjoyment, then, are four of my favorite systems papers.

Checking System Rules Using System-Specific Programmer-Written Compiler Extensions

Aside from the awful title, this is a fantastic paper. The authors managed to automate (in a practical, easy-to-understand manner) checking for common, simple bugs such as forgetting to drop a lock on error exit from a function. They found many bugs in widely-used real world systems with relatively little human effort. I've seen their stuff in action; it's good. And as usual for an Engler paper, it's well-written and a pleasure to read.

Sensing User Intention and Context for Energy Management

Don't you hate it when your laptop's screensaver starts up in the middle of a presentation? Angela Dalton and Carla Ellis explore using low-power sensors to make power-saving smarter. Angela had a great demo at HotOS: a laptop with attached camera that turned off the display when no

CALENDAR

« August 2007

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Today

RSS FEEDS

XML
All
/Operating Systems

SEARCH

Search

NAVIGATION

blogs.sun.com
Weblog
Login

REFERERS

Today''s Page Hits: 16

human face was visible. All day long, I saw people sitting in front of a laptop and then suddenly ducking to the side, again and again, testing her demo. I like that they are taking a creative new approach to power-saving, instead of fiddling with timeouts and mathematical models.

The Utility of File Names

Lots of research has been done with the idea that users will annotate files with performance hints using some obscure system-specific interface (yeah, right!). In this paper, Daniel Ellard shows that programs are already giving the system useful performance hints - by the names they give to files. Even more exciting, filename patterns and usage patterns can be automatically correlated by a modeler program.

The Google File System

I don't really need to describe this paper, do I? I just wanted to explain why I like this paper so much. Many people have tried write a distributed file system that is correct, generic, and performs well in all cases - and are still trying. What I like about GFS is that they picked a very specific problem (large scale distributed processing using a queuing system) and solved it in a specific way.

I hope you enjoyed the Confessions; thanks for reading!

(2004-09-21 15:27:46.0) Permalink

TUESDAY SEPTEMBER 14, 2004

ZFS on sun.com!

ZFS is the front page story on http://www.sun.com today:

ZFS--the last word in file systems

It's an excellent and accurate article.

Also, I turned off comments for now, after getting some blogspam. Email me with your comments and I'll respond here.

v a l . h e n s o n (a t) s u n (d o t) c o m

(2004-09-14 16:16:43.0) Permalink

THURSDAY SEPTEMBER 09, 2004

ZFS FAQs, FREENIX CFP, new systems paper

Today's post is a bit long, but I promise you'll find the entire post interesting even if you are only interested in reading about ZFS. Also, apologies in advance for the wacky fonts - the defaults don't seem to work quite right.

FREENIX CFP

FREENIX is coming! Submit to FREENIX! I'm especially encouraging all of you userland/application types to submit papers. FREENIX is intended to publish work on open source projects that isn't being published anywhere else, and while most kernel programmers think that the most trivial snippet of code is worth publication, userland programmers can easily write wildly popular million-lines-of-code systems with nary a thought about publishing. The deadline for submissions is October 22, 2004.

Cool systems paper: WAFL

The cool systems paper for today is File System Design for an NFS File Server Appliance, by Dave Hitz, et al. This paper describes the WAFL file system (WAFL stands for Write Anywhere File Layout), used internally by NetApp filers. In my opinion, this paper describes the most significant improvement in file system design since the original FFS in 1978. The basic idea behind WAFL is that all data is part of a tree of blocks, each pointing to the block below it. All updates to the file system are copy-on-write - each block is written to a new location when it is modified. This allows easy transactional updates to the file system.

The WAFL paper is a prime example of the kind of paper I'd like to see published more often (N.B. It was published in the Winter 1994 USENIX conference). From an academic standpoint, the paper is unacceptable due to style and format. From the standpoint of great new ideas, full implementation, and advancing the state of the art in practical file system design and implementation, it's a gem. Unfortunately, some people conclude that because NetApp filers use NVRAM to get acceptable (nay, excellent!) performance while holding to the NFS standard, the design ideas behind WAFL aren't useful for general purpose UNIX file systems. I say they're wrong - but read the paper and form your own opinion. The ZFS team thinks that a copy-on-write, transactionally updated general purpose UNIX file system is not only feasible but an excellent idea - which is why we wrote ZFS.

ZFS FAQs

Matt Ahrens, one of the primary architects and implementors of ZFS, stepped up to the plate and wrote about ZFS in his blog. Read Matt's blog entry introducing ZFS for a simple introduction to ZFS. Reading today's cool systems paper will also help you understand ZFS, since the basic philosophy behind some parts of WAFL and ZFS is similar. I'll add to what Matt has written and answer some of the most common questions people have asked me about ZFS.

Q. ZFS is just another dumb local file system. What is new about ZFS?

A lot of things! I'll try to hit the high points.

- No more partitions. Many ZFS file systems share a common **storage pool**. The analogy is that ZFS makes the sysadmin-to-

storage relationship like the programmer-to-memory relationship - you don't care where the storage is coming from, you just want storage. (Obviously, some administrators do care - and ZFS allows you to create many different storage pools if you need different kinds of storage for different file systems.) The administrator may add and remove disks from the storage pool with simple one-line commands, without worrying about which file systems are using the disks. The era of manual grow-and-shrink of file systems is over; ZFS automatically grows and shrinks the space used by a file system in the storage pool.

- The on-disk state is always valid. No more fsck, no need to replay a journal. We use a copy-on-write, transactional update system to correctly and safely update data on disk. There is no window where the on-disk state can be corrupted by a power cycle or system panic. This means that you are much less likely to lose data than on a file system that uses fsck or journal replay to repair on-disk data corruption after a crash. Yes, supposedly fsck-free file systems already exist - but then explain to me all the time I've spent waiting for fsck to finish on ext3 or logging UFS - and the resulting file system corruption.
- All data is checksummed. Even the checksums are checksummed! Between our checksums and copy-on-write, transactional updates, the only way the on-disk data can be accidentally corrupted without being detected is a collision in our 64-bit checksums (or a software bug - which is always true). With most other file systems, the on-disk data can be silently corrupted by any number of bugs - hardware failure, disk driver bugs, a panic at the wrong time, a direct overwrite of the disk by the administrator...

These are only the top three features of ZFS. ZFS has a million nifty little features - compression, self-healing data, multiple (and automatically selected) block sizes, unlimited constant-time snapshots - but these are the biggies.

Q. Why isn't ZFS a clustered/multi-node/distributed file system? Isn't the local file system problem solved?

Speaking from around 8 years of system administration experience, I can say that the local file system problem has most emphatically not been solved! Whenever someone asks me this question, I have to wonder if they ever ran out of space on a partition (especially frustrating on a disk with a lot of free space in other partitions), damaged their root file system beyond repair by tripping on the power cable, attempted to use any volume manager at all, spent a weekend (only a weekend if you are lucky) upgrading disks on a file server, tried to grow or shrink a file system, typed the wrong thing in /etc/fstab, ran into silent data corruption, or waited for fsck to finish on their supposedly journaling (and fsck-free) file system. Between me and two or three of my closest friends (none of whom are even sysadmins), we have run into all of these problems within the last year, on state of the art file systems - ext3, VxFS, logging UFS, you name it. As far as I can tell, most people have simply become accustomed to the inordinate amount of pain involved in administering file systems.

We're here to say that file systems don't have to be complex, fragile,

labor-intensive, and frustrating to use. Creating a decent local file system turned out to be more than big enough of a problem to solve all by itself; we'll leave designing a distributed file system for another day.

Q. What is ZFS's performance? Can I see some ZFS benchmarks?

ZFS is still under development, and the benchmarks numbers change day by day. Any benchmark results published now would only be a random snapshot in time of a wildly varying function and not particularly useful for deciding whether to use ZFS the released product. However, we can tell you that performance is secondary only to correctness for the development team and we are evaluating ZFS in comparison with many different file systems on Solaris and Linux. We can also tell you about some of the architectural features of ZFS that will help make ZFS performance scream.

- Every file system can use the I/O bandwidth of every disk in the storage pool - and does, automatically. We call this dynamic striping. Unlike static striping, it requires no extra configuration or stripe rebuilding or the like - ZFS simply allocates a block for a particular file system from whichever disk seems best (based on available space, outstanding I/Os, or what have you).
- Copy-on-write of all blocks allows much smarter write allocation and aggregation. For example, random writes to a database can be aggregated into one sequential write.
- Automatic selection of block size on a per-file basis allows us to choose the most efficient I/O size while making reasonable use of disk space.
- Integration of the volume manager and file system improves performance by giving more information than "write this block here and that block there" - the classic block device interface. Greg Ganger has a tech report discussing the problems of the narrow block device interface at the boundary between the storage device and the operating system; existing volume managers only compound the problem by introducing a second, identical block device interface where the information gets lost all over again. In ZFS, the "volume manager" - really, the storage pool manager - has all the information about, e.g., dependencies between writes, and uses that information to optimize performance.

Q. Isn't copy-on-write of every block awfully expensive? The changes in block pointers will ripple up through the indirect blocks, causing many blocks to be rewritten when you change just one byte of data.

If we only wrote out one set of changes at once, it would be very slow! Instead, we aggregate many writes together, and then write out the changes to disk together (and very carefully allocate and schedule them). This way the cost of rewriting indirect blocks will be amortized over many writes.

Feel free to ask more questions in the comments; I'll do my best to answer them.

(2004-09-09 20:29:37.0) Permalink Comments [5]

TUESDAY AUGUST 17, 2004

SHA-1 weakened, all other hashes broken I just finished watching the live webcast of the CRYPTO 2004 rump session, in which Eli Biham announced significant progress in finding collisions in SHA-1. He can do better than the birthday attack up to about 53 rounds (out of 80 total) of SHA-1 (this might be off by up to 10 rounds - I couldn't read the slides over the webcast). For reference, the time from a similar announcement for SHA-0 to a full collision was about 6 years - but most researchers were ignoring SHA-0 already as NIST proposed SHA-1 to replace SHA-0, due to known (to the NSA at least) weaknesses in SHA-0.

Eli Beham's talk was followed by announcements of full collisions in SHA-0 (by Antoine Joux), MD5, HAVAL-128, and RIPEMD (by Xiaoyun Wang). As a bit of fun, Xiaoyun Wang also presented a method to find collisions in MD4 that is so simple that it can be computed by hand (complexity 2^2-2^6 - that is, 4 - 64).

Before today, the state of the art in cryptographic hashes could be summarized as "Use SHA-1, everything else is either weak or unknown." Now it can be summarized as "SHA-1 is weak and everything else is broken."

I am, of course, ecstatic, as this strongly supports my paper opposing compare-by-hash, which depends on having a strong (not yet broken) cryptographic hash.

Thanks to Fred Douglis for adding a comment to my weblog pointing me to these results. For the record, no, I don't read Slashdot, but I'm beginning to think I should get back into the habit...

(2004-08-17 20:14:33.0) Permalink Comments [3]

THURSDAY JULY 22, 2004

The FREENIX track - has it succeeded too well? A few people mentioned that they find the papers I'm listing interesting, but they don't have enough time to read them because I'm listing them too quickly. C'mon, doesn't everyone have time to read a paper every two days? Fortunately, I'm at Ottawa Linux Symposium this week and I probably won't have time to introduce another paper for a few days.

OLS is my favorite conference, bar none, despite the fact that I started attending shortly after I stopped doing Linux development. OLS is a great conference at all levels - technical content, speakers, location, atmosphere, beer supply... Every year, each time slot has at least one talk or BOF that I'm interested in, and this year was a new high for quality in both talks and presentation. I think most of the excitement comes from the **relevance** of the work people are doing. Often, the code being presented is running on the laptops of half the people in the audience. OLS had a tendency in the past to present somewhat amateur rehashes of

design ideas already considered old-hat in the proprietary world, but that tendency has declined markedly. Most of the papers now focus on the real-world implementation of useful new functionality, with a thorough review of the roadblocks and the evolution of the design along the way. Rusty Russell's papers are a fine example of this style. Going to one of his talks gives you an realistic view of the way real world design and development proceeds - with many bad ideas, bugs, obstinate personalities, and wrong turns along the way.

The 700 pages (yes, 700) of proceedings (currently available as preprints) for this year's OLS led me to think about the purpose of the FREENIX track at USENIX Technical. FREENIX was intended to encourage publication of work going on in open source that wasn't, at the time, being published or recorded anywhere. FREENIX was to be a sort of gateway drug to get open source developers hooked on publishing, so the rest of the community could find out what was going on without trawling through hundreds of thousands of posts to linux-kernel. Hefting the two large tomes handed to me at the OLS reservation desk and comparing them to the rather more sparse FREENIX '04 proceedings, I had to wonder: Has FREENIX served its purpose? It certainly seems to me that a lot of publication is going on the open source world, as my aching shoulder can attest. Adding fuel to the fire, half of the USENIX general track papers use open source software (*BSD, Linux, Apache) as the base for experimentation.

At the same time that FREENIX seems less necessary, the USENIX general track has become more academically oriented, which is certainly not a bad thing. However, these two trends together are squeezing out what I consider the classic USENIX paper: the industry paper describing a useful new idea which is quickly adopted by the rest of industry, e.g., NFS or VFS or the slab allocator. While the trend towards more academic quality papers has resulted in more rigorous and thoroughly researched papers, I think we have paid a price. For a variety of reasons, papers about shipping products have a harder time fulfilling academic style requirements. An analogy may be the best illustration: This week's issue of "Science" (9 July 2004, Vol. 305, No. 5681) included a special section on immunology, in which one article ("Immunotherapy: Bewitched, Bothered, and Bewildered No More") decried several trends causing researchers to avoid human research; that is, research on actual, live human beings. One of the reasons they cited is as follows, from page 198:

> Many elite, basic science journals often consider valuable human studies to be inadequate because they lack the mechanistic depth we have come to expect from studies using laboratory organisms. Often precluded from publishing the best of human research in such journals, clinical scientists struggle for professional advancement, after having been excluded from the scientific mainstream. As editors ourselves, we now feel that basic research journals should get equally excited about hard-earned advances in understanding valid scientific problems as they exist in humans, trading some expectation of mechanistic insight for inherent relevance and respect for the demands of working within a proscribed lengthy human protocol.

To put it bluntly, research papers about humans can't be as thorough as those about mice because you're not allowed to "sacrifice" the human at the end of the study and examine their lymph nodes under a microscope, or implant wires into their brains without consent. Similarly, papers on open source or shipping software of any sort are under more constraints than, say, a paper about a research operating system. Standards compliance, stability requirements, backward compatibility, inability to get data from most customer sites, sheer time pressure - all contribute to a paper that doesn't fit the academic model. On the other hand, product-oriented papers can offer many things an academic research paper cannot - real-world feasibility, full implementation, long term experience, much wider user base, and more subtle revelations that only come with shipping a product to thousands or millions of users. I think there is a place for this kind of paper, and that place should be at USENIX Technical.

What, then, is to be done? I have a suggestion: Replace the general track and the FREENIX track with something akin to the "Product Track" and the "Academic Track" (terrible names, but you get the idea). The "Product Track" would be for software which is intended to be used by the masses. Examples include proprietary software products and a new scheduler for Linux someone is trying desperately to get merged. No one can force users to use a particular piece of software, so intent would be the key in classifying a paper as a "Product Track" paper. The "Academic Track" would be for blue sky research, or experiments on free software not intended to ever be merged back into the mainstream (most research being conducted by graduate students would probably fall into this bucket). Papers in this track would tend more toward the academic style. Ideally, all submissions would go into a common pool. Decisions on how to classify the papers and divide the work of reviewing them would be up to the conference organizers and the program chair.

I believe these tracks would encourage industry participation, preserve USENIX's hard-won reputation for academic excellence, and continue to encourage publication of work on open source software. Of course, reorganizing the tracks is not, by itself, a complete solution, but I believe it would be a step in the right direction.

(2004-07-22 17:22:54.0) Permalink Comments [1]

THURSDAY JULY 15, 2004

Peer-to-peer systems: Exposed! Some kind souls (Ted Leung and Bryan Cantrill) pointed out that my last name was nowhere present on this page; I have updated the description to be less witty and more informative. Of course, my full name is actually Valerie Aurora Henson, and if I used it in all its octosyllabic glory, that might clear up a common point of confusion about which pronoun to use when referring to me.

Today's snazzy systems paper is High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two, another 6-pager from HotOS IX. Surely you are familiar with all those peer-to-peer systems papers motivated by greed-tinged descriptions of gigabytes of unused disk space just waiting to be used on people's personal computers? Charles Blake was skeptical of a world where unused storage falls like manna from the

sky, so he created a few models and looked at some real-world data. It turns out that with realistic node membership times, member node bandwidths, and number of data copies necessary to provide reasonable availability, the amount of data that can be stored in a peer-to-peer network is severely limited by **cross-network bandwidth**, rather than available disk space. Blake puts it all so much more politely in his paper, with sentences like "We conclude that when redundancy, data scale, and dynamics are all high, the requisite cross-system bandwidth is beyond reasonable expectations."

This makes sense intuitively: How did most Gnutella users behave? They connected to the network long enough to download the songs they wanted, over their 56Kbps modem or 256Kbps cable modem, then left the network as soon as they were done. Most files were actually served by a relatively small number of reliable, high-bandwidth hosts; over a 3-day trace, the most reliable 5% of Gnutella peers provided 40% of the service. More pointedly, the authors estimate that the best 5% of Gnutella peers could be replaced by 10 university-run servers on cheap PC hardware - at which point the question becomes, "Why use many unreliable peers when a few reliable servers can serve as much data?" The answer is "When anonymity, security, or freedom from central control is important" - but never for performance or storage capacity reasons. In my opinion, this paper is a must-read for anyone working on peer-to-peer systems - but don't blame me if you switch fields shortly thereafter...

(2004-07-15 14:35:43.0) Permalink

TUESDAY JULY 13, 2004

Program committees, superpages, and compare-by-hash Today's entry is a triple whammy; hope you enjoy. I swear I'll write something about ZFS soon.

I have to agree with some of Werner Vogels' comments in the discussion about encouraging industry participation in USENIX:

> Having served on many program committees and having chaired a few also, I know that the hardest part of your job as PC chair is to get people from industry to join your committee. [...]
>
> People from product groups often do not have the time or the interest to join a program committee, and it does not get rewarded within the standard product development work practice, so it would be something they have to do in the evening hours.
>
> The same goes for paper writing by people from product groups, it is just not being done because there is no reward given within the enterprise for such an achievement (e.g. it will not not be a plus on your next performance review). A few papers from industry were written with support from managers, but in general they are volunteer efforts in the evening hours. [...]

> In my view it is not as much the program or steering committees that are the problem here. If industry really wants more exposure through conferences they can only do so through investing in their own organization. Make it easier for engineers to write papers and serve on committees, and reward them for doing this.

Why, Werner, were you spying on me during all those weekends I spent reviewing papers down at the local coffee shop?

I couldn't agree more, companies need to encourage and reward participation in program committees and writing papers. Unfortunately, the onus for change still falls on the engineers, as, in general, management doesn't see any connection between conference participation and revenue. I recently gave a talk at Sun on academic publishing and its relevance to Sun's bottom line; feel free to adapt my slides for presentation at your company (or your division of Sun).

- Slides (PDF)
- LaTeX source (needs papers.ps to build)

And for today's cool systems paper, I give you Practical, transparent operating system support for superpages. Superpages are also known as large pages or multiple page sizes; if you don't already know what I'm talking about by this point, I recommend just reading the abstract of this paper. I really liked this paper because it took what I regard as the correct approach to large pages: the decision about when to use large pages is done entirely in the operating system and requires no application modification **at all** (no cheating with silly linker tricks or launcher apps), and it improved the performance of a wide variety of everyday applications. The one part of the paper I dislike is the section on using compare-by-hash to detect dirty subpages; see my paper on compare-by-hash for why. Ironically, using compare-by-hash was a performance hit in this case; it frequently is in situations where the data is being compared locally. A draft paper I'm still working on discusses the performance tradeoffs in more detail.

(2004-07-13 17:28:35.0) Permalink Comments [10]

MONDAY JULY 12, 2004

Is your software crash-only? I've resisted starting a weblog for all the usual reasons - lack of time, neophobia, anti-herding instinct - but mostly because the only thing I really wanted to write was a rant about Sun's marketing strategy for ZFS (that's the Zettabyte File System, not Dynamic File Service, DFS, or DynFS), and management doesn't want us writing rants about Sun's marketing, no matter how entertaining they are. Finally, I had a halfway decent idea for a blog topic while talking to a presenter at USENIX '04: my favorite systems papers.

Crash-only Software

This is a short workshop paper that appeared in HotOS IX. From the abstract: "Crash-only programs crash safely and recover quickly. There is

only one way to stop such software - by crashing it - and only one way to bring it up - by initiating recovery." As motivation, the authors show that with a system running Red Hat 8.0 with ext3 as the root file system, it is faster to crash and recover (as in a power outage) than to cleanly shutdown and restart the system - 75 seconds to crash and recover versus 104 seconds for a clean reboot, with no "important" data loss in either case. (The irony of this result should be apparent to any systems person.) The authors argue that crash-only systems, which are made up exclusively of crash-only components, are a good choice for certain classes of problems, where the best way to deal with bugs is to simply restart (crash) the component behaving badly.

The crash-only philosophy is already widespread, most notably to myself as a file systems developer in Google's clustered file system, GFS (indeed, in all of Google's software), NetApp's WAFL file system, used internally in their filers, and ZFS, in which the on-disk state is always self-consistent. This paper simply clarifies the tradeoffs and properties of crash-only software, and, most importantly, introduces a nifty name for the concept. Recommended reading for all programmers. (2004-07-12 17:11:51.0)
Permalink Comments [3]